C advanced

Debugging

Richard Mörbitz, Manuel Thieme

Contents

It's not a bug...

There are different kinds of errors.

- Compilitime errors
- Runtime errors (*Bugs*)

Compiletime errors are easily handable since the compiler shows you where to fix them.

Bugs on the other hand are harder to find because you have no idea where to look for them.

... it's a feature.

Bugs can appear due to different reasons

- Variable overflow
- Division by Zero
- Infinite loops / recursions
- Range excess
- Segmentation fault
- Dereferencing *NULL pointers*
- ...

The dungeon

We prepared a little ASCII dungeon.
You can find it in *TODO: insert path*

- Look at the code and try to understand what should happen.
- If you find mistakes, please leave them. We'll fix them later.

- Copile it (with *-std=c99*)

- And now run it.

gcc flags

You can instruct *gcc* to display different warning levels and even abort compiling when warnings appear. These instructions are called flags.

-Wall	enables a bunch of compiler warnings
-Wextra	enables further compiler warnings
-Werror	warnings are interpreted as errors (compiling doesn't succeed)

- You can pass an arbitrary amount of flags
- Seperate them with whitespaces

Debugging light

- Compile the dungeon with *-Wall* and *-Wextra*
  - There are 3 Warnings. The 3rd one matters.
  - There is a variable that's set but not used.
- You may find a bug there. Fix it.

The GNU DeBugger

There are tools helping with bugs, called debuggers. GDB is one of them.
To Use it

- You have to install the package *gdb*
  On Windows, hope your cygwin installation came with it
- You have to compile your program with the *-g* flag

```
$ gcc -g main.c
```

- After that you can start your program with gdb:

```
$ gdb a.out
```

Commands

- If you started gdb without a file you can load it with *file file_name*.
- Use *run* to execute the program with gdb.
  You should begin with it. It will give you further information about the crash.
- You can set an arbitrary amount of breakpoints with *break line_number* or *break function_name*.
  Begin with a breakpoint at the point the program crashes.
- Print values with *print identifier*.
- Use *watch identifier* to break and print a variable when it's changed.

Once you're at a breakpoint

- Use *next* to execute the next program line only.
- You can jump to the next breakpoint with *continue*.
- To see How you have come to this point in the program flow, type *backtrace* or *bt*.
  This shows you all functions you called to come there.
- By only hitting the *return key*, you repeat the last entered command.

GDB is much more mighty than this few commands, but that should be sufficient to solv your final quest.

Now it's up to you

- Find and fix all Bugs in the dungeon.

file	load program
run	execute program
break	set breakpoint
print	print variable
watch	print variable when it changes
next	execute next line and break
continue	execute until next breakpoint
backtrace / bt	How did i end up here?